Plankton Classification Using ConvNets

Abhinav Rastogi
Stanford University
Stanford, CA
arastogi@stanford.edu

Haichuan Yu
Stanford University
Stanford, CA
haichuan@stanford.edu

Abstract

We present the conditions of a data science challenge and inspect the official training dataset provided. We discuss our technical approach, and address the challenge using ConvNets. We implement several techniques to improve the validation accuracy and reduce the loss. We compare results, visualize some of the error cases, and perform a detailed error analysis.

1. Introduction

1.1. About the problem

In this work, we investigate the problem of plankton classification using data provided by Booz Allen Hamilton through the Kaggle National Data Science Bowl.[1] The results of the Data Science Bowl would deliver a positive social impact for the worlds oceans, providing the key technology in a new instrument to assess ocean health, a task that has been unwieldy, inaccurate, slow, and expensive using current technologies involving manual plankton classification. This work will contribute to the community of work done for the data science competition.

The Kaggle Data Science Bowl evaluates solutions to the problem using the multi-class logarithmic loss function same as that of the softmax classifier. Regarding intuition, this number tells us how confident we are in predicting the class of an image. Optimally, the loss will be zero, where we correctly classify each image with full confidence, ($p_{y_i} = 1$). The loss formula, also known as the softmax loss, is:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\ln(p_{ij})$$

where ln is the natural logarithm, y_{ij} is an indicator function for whether image i is in class j, p_{ij} is the predicted probability for image i belonging to class j, M is the number of classes and N is the number of images.

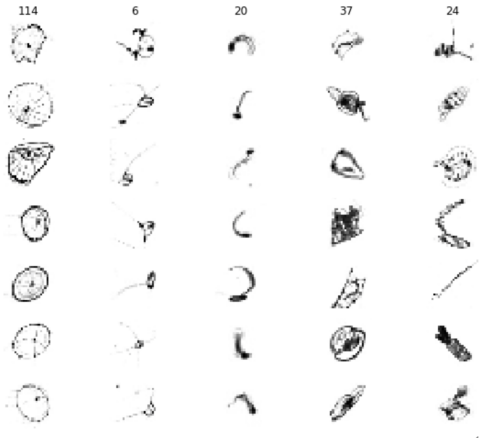

Figure 1. Several plankton images from randomly selected classes (columns) from the Kaggle Data Science Bowl Dataset

1.2. Description of dataset

We use the official data provided by the Kaggle Data Science Bowl. The dataset is comprised of approximately 30,000 labeled training images and around 130,400 unlabeled test images. Some of the training images are visualized in Figure 1. There exist 121 unique labels for the images. We do not use any outside data in either training or testing of our model. However, since the organisms in the image have any orientation in the image, we can augment our training data by rotating and/or reflecting the original image.

1.3. Challenges

The distinguishing features of the problem which make it interesting and difficult are-

- Very diverse dataset, ranging from the smallest single-celled protists to copepods, larval fish, and larger jellies.
- The organisms in the image can have any orientation within 3-D space.

- Some of the classes correspond to the same genus but different species and hence appear very similar.
- Few classes in the dataset correspond to unidentified planktons and objects.
- 12 classes have fewer than 20 images each in the training set, making it difficult to classify unseen examples
- Some images are too noisy to be unambiguously labeled by the experts. Some amount of noise in the training data is thus inevitable.

1.4. Expected Results

As part of the competition, we are required to submit a csv file listing the class probabilities of 121 classes for each test image. The submissions are evaluated based on multiclass softmax loss.

The current leader in the Kaggle Data Science competition, *Deep Sea*, currently has a test loss of 0.560994, (as of 8 PM on 3/15/15). The current leader's approach and implementation is currently unknown.

2. Background/Related Work

2.1. ConvNets

In recent years, Convolutional Neural Networks (ConvNets) have won the main image classification competitions in the computer vision community. ConvNets are well posed to solve the problem of image classification because of the basic neural network architecture and the parameter sharing which makes the features translationally invariant.

A neural network is comprised of many units inspired by biological neurons, that is, they are interconnected and hold weights and biases. The neural network can be trained such that the weights and biases are modified to learn an optimal set of numbers for a given task.

Each layer in a neural network architecture represents a linear function, and each successive layer enables the network as a whole to represent a higher order function. Therefore, complicated functions may be expressed by a deep network.

A ConvNet has the basic neural network architecture, but with four primary types of layers, convolution, rectified linear units (ReLU), pool, and fully connected (FC). With the labeled input images, the weights can be driven by a momentum supported stochastic gradient descent minimizing a loss function.

2.2. ConvNet Modifications

The classification error of a ConvNet can be reduced by several methods. Fine tuning, gathering more training data, and increasing the depth of the ConvNet are all possible approaches. Another approach is to modify certain properties

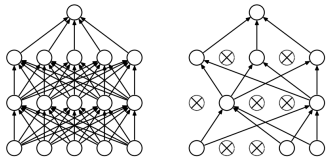

Figure 2. Visualization of a neural network (right) with dropout (left) [4]

of ConvNet layers. This subsection explores some of these modifications.

2.2.1 Dropout

Dropout is a method that reduces overfitting in ConvNets by randomly silencing neurons with a certain probability during training. This has a similar effect to training multiple sub-ConvNets within a larger ensemble ConvNet, and taking the best result. Srivastava et al showed that it improved validation accuracy on a variety of training datasets, meaning that it could potentially reduce overfitting on this dataset as well [4].

2.2.2 Data Augmentation

Another way to improve the performance of ConvNets is by simulating variations in test data. By looking at training data, some of the types of variations in test data can be expected. By making similar variations to our training dataset during training, we can improve the likelihood that our ConvNet will be able to correctly classify such edge cases.

Data augmentation is especially useful when training data is limited. For the Kaggle data science contest, we have very small training data sets for some classes, which means that we may receive new cases for these classes that the ConvNet has not seen before. In other classes, there exists a lot of intra-class variation, specifically with respect to the orientation of the subject. Across all classes, there also exists noise. These image irregularities may contribute to misclassifications of slight variants to the data.

We experiment with several kinds of data augmentations that we observe in our training data. Specifically we observe rotation and flips as useful augmentations to the training image. With these augmentations, the trained network should still have data integrity, but will also be more likely to correctly classify unseen classes.

2.2.3 Leaky ReLU

Leaky ReLU units have been shown to have a small but consistent improvement in classification accuracy in deep neural networks [6]. A recent paper that surpassed 'human' level performance on the ImageNet challenge used

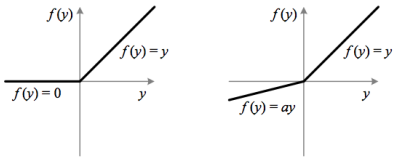

Figure 3. ReLU (left) and Leaky ReLU with a learnable gradient (right) [5]

a Leaky ReLU layer with a learnable gradient to optimize weight training [5]. By parameterizing the ReLU gradient as a quantity that can be updated, the ConvNet can converge more quickly.

2.2.4 Xavier Initialization

Weights and biases are typically initialized randomly. The code used in the assignment uses a random Gaussian distribution to initialize the code. The issue is summarized in the following:

"If the weights in a network start too small, then the signal shrinks as it passes through each layer until its too tiny to be useful. If the weights in a network start too large, then the signal grows as it passes through each layer until its too massive to be useful. Xavier initialization makes sure the weights are just right, keeping the signal in a reasonable range of values through many layers." [7,8]

Therefore, we could consistently expect the same or improved results when training a ConvNet when working with a Xavier initialization.

3. Approach

Because of a large number of classes and high variability in the dataset, ConvNets seem to be a very good choice for Plankton Classification. The competition tutorial on Kaggle demonstrated classification using the aspect ratio of images contained in the dataset. This achieved a training accuracy of 44.61% and a validation loss of 3.74 using 5-fold cross validation. This motivated us to examine the performance of Non-ConvNet approaches

3.1. Non-ConvNet approaches

1. **Major-minor axis ratio of convex hull** After examining the datasets, we noticed that members of some of the classes are round, some are elliptical and some are relatively thin and long. So, we calculated the convex hull of the pixels belonging to plankton in the image and used the the ratio of the major and minor axis of the convex hull as a feature. A two layer neural network was trained on these features. Our best hyperparameter setting achieved a training and validation accuracy of 13%, which was not impressive.

2. **Raw pixels and neural network** The pixel intensity of the images resized to size 24x24 were used as the inputs to a two layer neural netwok. We held out one-sixth of the training data for cross validation. Our best network achieved training and validation accuracy of 29.70% and 28.14% respectively and achieved a loss of 3.0255 on the test data.

3.2. ConvNet based approach

The other participants of the competition had already reported good results using ConvNets. We experimented with various architectures, regularization and augmentation techniques using ConvNets.

1. **Two layer ConvNet baseline** - We trained a two layer ConvNet as a baseline to gauge the performance of ConvNets. In all our ConvNet approaches, we have retained one-sixth of the training data for cross-validation. Thus our training set consists of 25,000 images and the validation set consists of about 5,000 images, randomly chosen from the given labelled data. Our best network, using 60 channels of 5×5 filters, achieved training and validation accuracy of 59.90% and 49.77% respectively and a loss of of 1.937 on the test set after training with dropout.

2. **Three layer ConvNet**- We gradually increased the complexity of our model hoping to achieve a better performance. Hence, we added a fully connected layer before the softmax layer, consisting of 64 affine units. We kept the first layer as a 3×3 conv layer with 32 filters. This model was found to be overfitting, giving us a training and validation accuracy of 72.70% and 52.28% respectively, achieving a loss of 1.713 on the test set. We didn't spend time on tuning the hyperparameters of this model because we were hoping to achieve a better performance using a deeper model.

3. **Five Layer ConvNet** We used the same network implementation provided in the CS 231N, Assignment 3. Our data consists of grayscale images. We replicated the grayscale image across the three channels to get a RGB representation. The first three layers of the network are Conv-ReLu-Pool layers having a filter size of $5 \times 5 \times 3$, with 32, 32 and 64 channels respectively. We do 2×2 max-pooling in all these layers. After tuning the hyperparameters of the network, we obtained a training and validation accuracy of 79.10% and 63.79 % respectively obtaining a loss of 1.258 and 1.427 with and without dropout respectively. Among all our networks, this network gives the lowest loss on the test data.

4. **Six Layer ConvNet** We implemented a six layer ConvNet using the CS 231N Assignment 3 framework. After many tries, we were unable to train this network using random weight initialization. We noticed that the architecture of the first 4 layers of this network is same as that of the five layer network. After transferring the weights learned from the 5 layer network, we were able to get it to train. Using all the weights from the smaller network was not giving good results, with the 6 layer network achieving a lower validation accuracy than the 5 layer network.

   We experimented with selectively transferring as few weights as possible and getting the network to train. We could not get the network to train if we only transfer the biases of the first four layers and initialize the other weights randomly or we only transfer the weights of 2nd, 3rd or 4th layer. However, we could get the network to train by only transferring the weights of the first layer. The trained network achieved a validation accuracy of 66.06%, which is slightly better than the original network. The achieved training accuracy was 79.60%. However, the loss on the test set was 1.464, higher than the five layer network.

3.3. Evaluation Metrics

We use Validation Accuracy and the Overfitting Ratio as complementary metrics to understand how to improve our ConvNet during training. Validation Accuracy is given as:

$$ValAcc = \frac{\#ValidationImagesCorrectlyLabeled}{\#ValidationImages}$$

And the Overfitting Ratio is:

$$ORatio = \frac{TrainingAccuracy}{ValidationAccuracy}$$

Where the Training Accuracy is:

$$TrainAcc = \frac{\#TrainingImagesCorrectlyLabeled}{\#TrainingImages}$$

By considering the overfitting ratio during training, we can know the current state of the model and how to modify the ConvNet to better train it, which might be to add more layers, increase the number of filters, or other modifications.

4. Experiments

4.1. Improving the Five Layer Convnet

We did various experiments to decrease the loss on test data. They are-

Layers	Filter Size	Val. Accuracy	Val. Loss	ORatio
5	3 x 3 x 3	58.11%	1.749	1.09
5	3 x 3 x 7	58.51%	1.592	1.18
5	3 x 5 x 7	58.73%	1.584	1.19
5	5 x 1 x 5	$\leq$ 10% *	-	-
5	5 x 3 x 5	58.30%	1.887	1.18
5	5 x 5 x 5	63.79%	1.258	1.24
5	5 x 5 x 5**	63.97%	1.242	1.27

Table 1. 5-Layer Network Models trained with varying filter sizes over 15,000 updates without data augmentation. * Terminated after 10 epochs, low performance. **Trained for 75,000 updates

1. **Varying filter sizes in the first three conv layers**

   We experimented with various filter sizes, and the results are shown in Table 1. Looking at these results, we found that the optimal filter sizes were 5x5 for all three layers. After exploring the space, we trained the model with filter size 5 x 5 x 5 for 75,000 updates, and found a negligible improvement (0.16%) in validation accuracy and a small increase in overfitting ratio (0.03). Without further modification to the ConvNet, this may be the best 5 layer model in this paper.

2. **Not doing max-pooling in some conv layers** Max-pooling reduces the size of the volumes and hence the number of parameters. Our best network had max-pooling in all the three conv layers. We experimented with two different variations -

   (i) Turning off pooling in first two layers- This increases the number of parameters in the 4th FC layer by about 16 times. The network was found to train very slowly and showed large overfitting after 10 epochs.

   (ii) Turning off pooling in the first conv layer- This increases the number of parameters in the 4th FC layer by about 4 times. Although this network was not showing large overfitting, it achieved a training accuracy of 76.40% and validation accuracy of 61.72% after training for 60 epochs with a higher value of regularization(0.005)

3. **Data augmentation by random rotation and flips** Data augmentation comes as a natural step because of the very nature of our dataset. Our dataset contains images which can correspond to an arbitrary 3D orientation of the plankton. We have done data augmentation during the first 80% of epochs of training by doing horizontal and vertical flips and rotations of the multiple of 45^o. Data augmentation for an image is done with a probability of 0.5 by applying a randomly chosen operation from the ones listed above. Classifier trained

using data augmentation had very less overfitting but the performance was similar to the networks trained without augmentation, achieving a training and validation accuracy of 67.80% and 65.16% respectively.

4. **Application of Leaky ReLU and learnable gradients**

Leaky ReLU	Learning	Val. Acc.	Val. Loss
Off	Off	52.62%	2.21
On	0.1	47.39%	1.85
On	0.1	46.60%	2.47
On	0.25	36.11%	2.27

Table 2. 5 Layer Network Models with filter sizes 5x5x5, tested with and without leaky ReLU and learnable leaky ReLU, trained over 5,000 updates.

While the literature suggests that we should obtain better classification accuracy with leaky ReLUs, our experiments suggested otherwise because of the lower validation accuracy (52.62% vs 47.39% without and with leaky ReLU respectively). In the case of a Leaky ReLU without a learnable gradient, however, the Validation loss was found to be significantly lower than in the case without a Leaky ReLU (1.85 vs 2.21 with and without Leaky ReLU respectively).

As Kai et al suggested an initialization of the learnable gradient at 0.25, we did so as well, and observed a similar loss to the result without Leaky ReLU (2.27 vs 2.21) [5]. However, in this case, the validation accuracy was lower by a significant margin of 16%. A simple conclusion to draw would be that by parameterizing the Leaky ReLU gradient to be learnable is causing overfitting on non-essential features, however, interestingly, the overfitting ratio for the model was 0.96. This implies that the model has not yet been overfitted, but it is not clear whether more training iterations would cause the model's validation accuracy to improve. It is possible that with more training iterations and more fine tuning, a better validation accuracy and loss could be obtained.

5. **Using L1 norm for the FC layer and L2 for conv layer weight regularization** L1 regularization results in sparse weights whereas L2 regularization results in weights having evenly distributed components. Since Conv layer weights are shared across all the pixels, they contain relatively rich information and hence we don't want to make them sparse. Making the weights of the 4th layer(FC layer) sparse resulted in training and validation accuracy of 69.40% and 62.67% respectively.

6. **Xavier Initalization**

Xavier Init.	Leaky ReLU	Val. Acc.	Val. Loss
Off	Off	47.67%	1.91
On	On	6.63%	4.14

Table 3. 5 Layer Network Models with and without Xavier Initialization for leaky and non-leaky ReLUs after 5,000 updates

As predicted by Kai et al, using Xavier initialization leaky ReLU resulted in poor performance, not increasing above 6.63% in validation accuracy[5]. With more time, more experiments investigating the efficacy of Xavier Initialization would have been conducted.

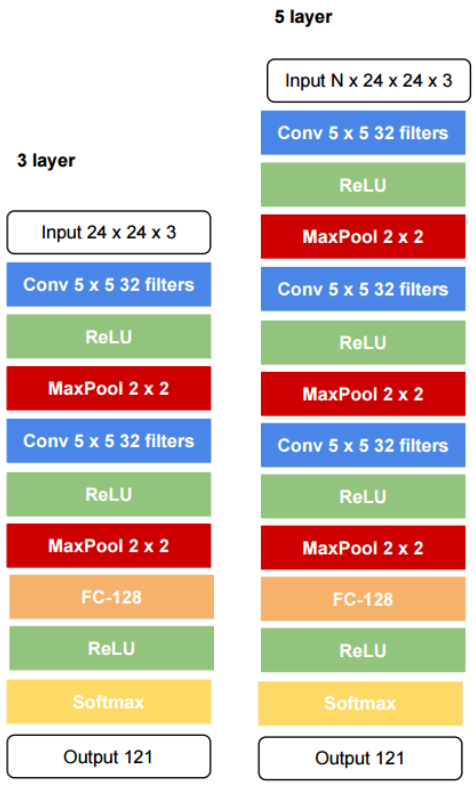

Figure 4. Three Layer and Five Layer architecture used in ConvNet

4.2. Analysis of Results

We have done experiments to visualize the working of ConvNets and error analysis to understand the strengths and weaknesses of our classifiers. In the following, we show the results obtained for the classifier achieving the lowest loss on the test data.

1. **Features extracted by ConvNet** Since ConvNets perform so well, the features extracted by them are worth

investigating further. To visualize the high dimensional features, we use a plotting tool called t-SNE, which projects high dimensional data into a low dimensional space so that the relative distances are not distorted much.

Using this tool, we generated plots for raw pixels and features extracted by ConvNet for 250 randomly chosen images from 10 randomly chosen classes. It can be observed that features extracted by ConvNets are more discriminative and hence perform better in classification.

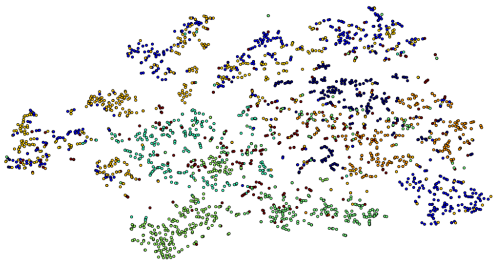

Figure 5. t-SNE plot of raw pixels for 250 images from 10 classes

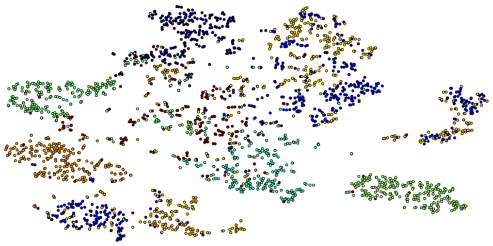

Figure 6. t-SNE plot of features extracted by Convnet for same images used in Figure 4

2. **Most confused classes** In our best classifier, we investigated which classes were being misclassified the most. We visualize 7 randomly sampled images from top 5 misclassified classes from the training set and validation set in Figure 7 and 8 respectively. The intra-class variability in these images is quite apparent even when we have just sampled 7 images from each class, which might be the cause of the poor performance.

3. **Relationship between misclassification and data** The provided dataset is highly skewed in terms of number of training images for each class. The average number of training and validation images is 250 per class but 12 classes have fewer than 20 images in training and validation set combined. This made us wonder whether there was some relation between the classification accuracy and the number of images for a class in the dataset.

Figure 7. The top 7 mislabeled classes in the training data

Figure 8. The top 7 mislabeled classes in the validation data

Furthermore, the top 7 most misclassified classes in training set have 25, 9, 15, 9, 9, 17 and 12 images in the training set. So it is possible that these images are getting misclassified because of the lack of data. However, the top 7 misclassified classes in the validation set have 152, 115, 40, 45, 38, 44 and 46 images in the training set respectively. So there seems to be enough data to train these classes well but the classifier doesn't generalize well to unseen examples. This might be due to high intra-class variability of the images.

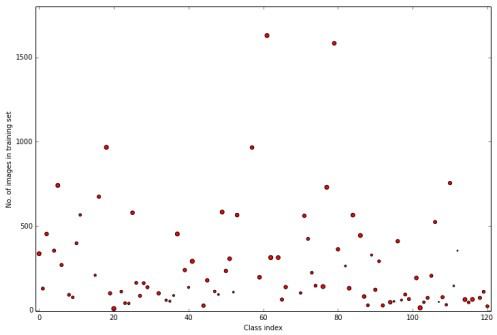

Figure 9. Relationship between validation accuracy and number of images in training set for each of 121 classes. The size of points is proportional to validation accuracy

The scatter plot in Figure 9 shows the number of training examples (y-axis) for each class(x-axis) in the training set. The size of the points is proportional to the validation accuracy for that class, defined as the number of correctly classified images of a class in the validation set divided by the total number of images for that class in the validation set. Small bubbles are only present in the lower half of the plot but large bubbles are present throughout. So we can say that more data guarantees good accuracy in our case. However some classes show a good accuracy even with relatively small data.

4. **Most confused class pairs** Another good visualization for the performance of a classifier is to quantify which classes confuse it the most. For a classifier $\mathcal{C}$, the confusion between two classes m and n can be defined as -

$$\text{Conf}_{\mathcal{C}}(\text{m}, \text{n}) = \frac{\text{\# images in class m labeled as n by } \mathcal{C}}{\text{\# images in class m}}$$

The values of pairwise class confusion can be visualized by the image in Figure 10. The intensity of a pixel in ith row and jth column is proportional to $\text{Conf}_{\mathcal{C}}(i, j)$. The dominant diagonal tells us that classification accuracy is fairly good for most of classes. The breaks in the diagonal indicate the classes which are being confused the most with some other class. Example images from the 5 most confused class pairs can be seen in Figure 11.

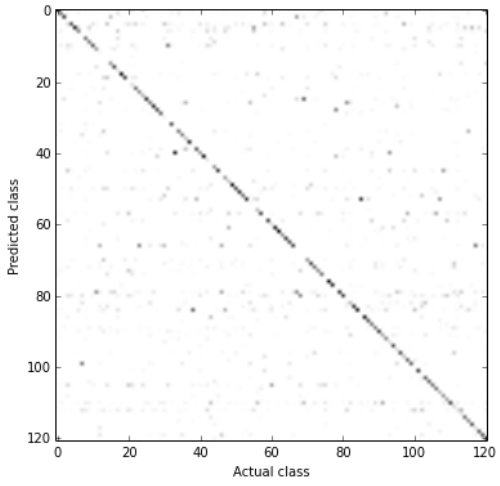

Figure 10. Image showing confusion for class pairs. Darker pixels correspond to higher value of confusion

5. Conclusion

We undertook a fairly difficult classification problem in this project, which is challenging due to presence of large number of similar classes and relatively small data. We examined the effectiveness on ConvNets in extraction of features from these images and trained a five layer ConvNet achieving a loss of 1.25 on the test data and a validation accuracy of 65.16%. We did various experiments to improve the performance of ConvNets and also did a detailed performance analysis.

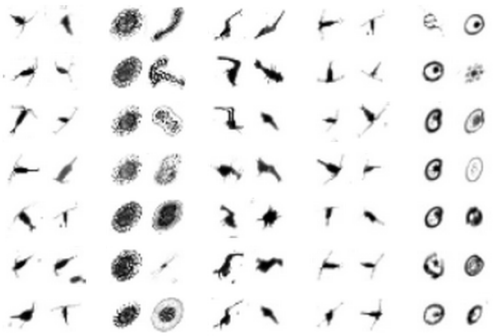

Figure 11. Most confused class pairs. Each column of two images consists of 7 randomly chosen images for the most confused class pairs ordered from left to right in decreasing order of confusion. The class of the left image is being confused as the class of right image

References

1. **National Data Science Bowl**
   http://www.kaggle.com/c/datasciencebowl
2. **CS 231N Lecture Notes**
   http://cs231n.github.io/
3. **CS 231N Lecture Slides**
   http://cs231n.stanford.edu/
4. **Dropout: A Simple Way to Prevent Neural Networks from Overfitting**
   http://www.cs.toronto.edu/ rsalakhu/papers/srivastava14a.pdf
5. **Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification**
   http://arxiv.org/pdf/1502.01852.pdf
6. **Rectifier Nonlinearities Improve Neural Network Acoustic Models**
   http://ai.stanford.edu/amaas/papers/relu_hybrid_icml2013_final.pdf
7. **An Explanation of Xavier Initialization**
   http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization